REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
		FINAL/30 SEP 91 TO 31 MAR 95

4. TITLE AND SUBTITLE
DEDUCTIVE DATABASES AND KNOWLEDGE BASE SYSTEMS

5. FUNDING NUMBERS
2304/GS
AFOSR-91-0350

6. AUTHOR(S)
PROFESSOR JACK MINKER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
DEPARTMENT OF COMPUTER SCIENCE AND
INSTITUTE OF ADVANCED COMPUTER STUDIES
UNIVERISTY OF MARYLAND
COLLEGE PARK, MARYLAND 20742

8. PERFORMING ORGANIZATION REPORT NUMBER
AFOSR-TR- 95-0455

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR/NM
110 DUNCAN AVE, SUTE B115
BOLLING AFB DC 20332-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFOSR-91-0350

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE: DISTRIBUTION IS UNLIMITED

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The following sections describe the research that has been accomplished during the entire grant period form September 1, 1991 to March 31, 1995, with partial support from the Air Force Office of Scientific Research and summarizes the papers, prototype systems and theses produced on the grant. Several major areas will be discussed. These are **Cooperative Databases Systems, Combining Knowledge Base Systems, Disjunctive Deduction Databases, Extensions to the Semantics of Logic Programming and Parallel Logic Programming**

During the first year of the grant we emphasized theoretical aspects of work in deductive databases and knowledge base systems. During the second year we stressed the development of tools and techniques to transfer the theory into practical systems. During the third year we developed prototype systems and experiments with them. Technological developments were made in Cooperative Database Systems and Disjunctive Deductive Databases. New theoretical developments were achieved in the Semantics of Logic Programming and in Combining Databases.

14. SUBJECT TERMS

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR(SAME AS REPORT)

FINAL PROGRESS REPORT: RESEARCH IN DEDUCTIVE DATABASES and KNOWLEDGE BASE SYSTEMS GRANT NUMBER AFOSR 91-0350 September 1991 - March 1995

Jack Minker, Principal Investigator
Department of Computer Science and
Institute for Advanced Computer Studies
University of Maryland
College Park, Maryland 20742

1 Introduction

The following sections describe the research that has been accomplished during the entire grant period from September 1, 1991 to March 31, 1995, with partial support from the Air Force Office of Scientific Research and summarize the papers, prototype systems and theses produced on the grant. Several major areas will be discussed. These are:

- *Cooperative Database Systems.*
- *Combining Knowledge Base Systems.*
- *Disjunctive Deductive Databases.*
- *Extensions to the Semantics of Logic Programming.*
- *Parallel Logic Programming.*

During the first year of the grant we emphasized theoretical aspects of work in deductive databases and knowledge base systems. During the second year we stressed the development of tools and techniques to transfer the theory into practical systems. During the third year we developed prototype systems and experimented with them. Technological developments were made in *Cooperative Database Systems* and *Disjunctive Deductive Databases.* New theoretical developments were achieved in the *Semantics of Logic Programming* and in *Combining Databases.*

Parallel logic programming is a topic supported by the AFOSR on a previous grant. The work was continued with support from the National Science Foundation. We report upon the final results of this research as it was initially supported by the AFOSR.

Work on the semantics of logic programming was supported almost entirely by the National Science Foundation. Part of Jack Minker's time was spent in developing a monograph, **Foundations of Disjunctive Logic Programming** [LMR92]. The work on the monograph is directly related to the efforts in disjunctive deductive databases. We briefly summarize some of the results on logic programming that were partially supported by the AFOSR.

2 Research Summary

2.1 Cooperative Database Systems.

2.1.1 Overview of Cooperative Query Answering Work

The objective of the work in cooperative answering systems is to provide answers to users that respond correctly to a query, account for users' needs, and inform users of reasons underlying the response to queries. The state-of-the-art in cooperative answering has been to develop *ad hoc* tools for particular databases. Our work has been oriented towards the development of tools that are general and that can be transferred to different databases in a straightforward manner. While the techniques are general, they use information that should exist within any realistic relational or deductive database. The information we use from a

relational database are its integrity constraints and view definitions. In deductive databases, rules allow new data to be derived, and we use the integrity constraints and rules already present to provide cooperative responses.

Principles of cooperation can be applied successfully to knowledge base systems built within the deductive database and logic programming paradigm. We have been looking at how to better meet a user's needs within the framework of the cooperative answering system of Gal and Minker [GM90*]. The basic system uses semantic information in a database to provide augmented responses to users.

In [GGM91*], we introduced a cooperative method called *relaxation* that expands the scope of a query by relaxing constraints implicit in the query. The method exploits the semantic network-type information inherent in the rules of a deductive database, changing constants to variables and changing predicate letters to generate meaningful relaxed queries to offer to the user. An expanded version of this paper [GGM92b] was published. An overview and description of the work in cooperative answering that has been taking place at the University of Maryland up to 1992 is presented in [Gaa92a]. Based upon our work in cooperative answering systems, we were invited to write a survey article on cooperative answering systems. In [GGM92a] we present an overview of work that has been performed on this topic up to 1992. We discuss foundational work in cooperative answering systems that has been achieved on the topics of beliefs and expectations, presuppositions, misconceptions, generalizations, intensional answers, and user models.

In addition to the work in completing the above papers, we have focused on transferring the technology we have been developing on cooperative systems to full-fledged cooperative database systems which can be interfaced with current relational database systems such as *INGRES'89* (also known as *University INGRES*). We developed an architecture for a cooperative answering system [GGMN92c, GMN94] and revised our prototype system to conform to the architecture. We also developed efficient algorithms that are important for an operational system. We have been engaged in developing efficient algorithms that render the search for cooperative query answers tractable, which is necessary for a functional cooperative database system [Godf94, Godf95]. The system was integrated with *INGRES'89* and tested on a database that consists of information about naval ships. Experimentation with the database and the cooperative system indicates that the overhead in using the cooperative shell over a relational database was not significant and that the approach was viable. Based upon our research, we believe that it will be possible to incorporate a cooperative answering mechanism with virtually any database management system at a small cost in overhead.

During the last few months of the grant, we obtained a copy of the *ORACLE-7* database system, a production database system (*INGRES'89* is not a production database system). We have started to integrate the *Carmin* cooperative engine with *ORACLE-7*. Assuming that the system works well, we plan to make it available to the AFOSR, although we will be completing it with other funding.

Some work was also accomplished on *natural language generation of cooperative answers.* A model of natural language generation that incorporates theories of tense and aspect for the generation of coherent text from a temporal database is presented in [DG92]. Procedures have been devised to select tense and aspectual features that are crucial for the surface realization of natural language utterances from the logical expressions of a temporal database. A procedure for selecting temporal connecting words for complex matrix/adjunct sentences

that describe two conjoined database literals has also been built. The approach to selecting tense, aspect, and connecting words is a general method to handle temporal information in the generation of language. The current framework has been applied directly to the task of generating cooperative answers and to the language generation task in machine translation.

A Ph.D. thesis [Gaa92b] was completed on cooperative answering. A second Ph.D. thesis will be completed before the end of December 1995 [Godf95].

2.1.2 The Carmin Prototype

In this section we describe the *Carmin* prototype which can be made available for beta-testing and outside use. *Carmin* is a cooperative database system. The primary cooperative response features that the *Carmin* project covers are:

- recognizing misconceptions,
- recognizing minimal failing subqueries (false presuppositions), and
- query relaxation.

Misconceptions arise when a query must fail with respect to the semantics (rules and integrity constraints) of the database. *Carmin* generates an explanation for the user in such cases. The misconception engine and explanation facility are implemented in the prototype.

The approach [Godf94] and code for minimal failing subqueries have been accomplished as part of the *Carmin* project, but not yet integrated into the *Carmin* prototype. We plan to explore adding a query relaxation mechanism to the prototype. This mode will require more consideration and research. A previous prototype implements query relaxation, but this is missing in the current prototype. We still need to determine a principled way to add this to the current prototype, and how to integrate it appropriately with the other features.

The paper [GMN94] gives a good overview of *Carmin*'s intended cooperative response modes and of *Carmin*'s architecture. This paper will also provide more details about what these cooperative response modes are, along with examples.

Carmin's query interface is *Datalog*. In the case of *INGRES'89*, the query is translated internally to a set of "SQL" queries that are then sent to the associated relational database system for evaluation. *Datalog* is a higher level query language than SQL, so this translation can be involved. *Carmin* has a sophisticated *Datalog-to-SQL* translator that results in good query evaluation performance. (*Carmin* is a simplistic *deductive* database system.) See the paper [GMN94] again for some more details on this.

Some people may find using *Datalog* easier than SQL, since it is a higher level query languages. Others may not; especially trained SQL programmers who are very used to SQL. *Carmin* must use the *Datalog* format internally, as it must do a logical analysis of the query. *Datalog* is amenable to this, but SQL is not so readily. It may be possible in the future, though, to add an SQL front-end to *Carmin* that would allow it to accept and process SQL-like queries. Whether we pursue this will depend on feedback from our various beta-testers.

The current *Carmin* prototype accomplishes a limited degree of semantic query optimization. We shall be working to add further semantic query optimization techniques to future versions of the *Carmin* prototype.

A slightly older version of the *Carmin* prototype (with a less sophisticated explanation facility) is available as a demonstration on World Wide Web for evaluation. The demonstration pages give a more detailed overview of *Carmin*. The demonstration is available at:

URL = http://karna.cs.umd.edu:3264/

The *Carmin* system is implemented in Quintus Prolog and C. The beta-site organization would need Quintus Prolog to run the system. Quintus is a commercial Prolog system that must be licensed. It may be possible to convert the code to *Sicstus or SWI Prolog*, although this would require nontrivial effort. We run the system on a SUN SPARCstation under SUN OS UNIX. The system should not be operating system dependent, and ought to be easily ported to other platforms.

The current *Carmin* system is interfaced with *INGRES'89*, a free-ware relational database system made available from Berkeley for educational and research purposes. *Carmin* can be used with databases in *INGRES'89*. We plan to implement a second RDBMS interface, this with *ORACLE-7*, the newest commercial version of the Oracle Company's main relational database system.

2.2 Combining Knowledge Base Systems.

When dealing with multiple knowledge base systems, the individual systems may be consistent with the integrity constraints, but when combined, the new system may be inconsistent. The problem of combining knowledge bases is studied with respect to three knowledge representation languages: *logic programs, first-order theories, and default theories.* The various properties that a combined knowledge base should satisfy are formalized as:

1. consistency,
2. maximality, and
3. correctness with respect to the union of the knowledge bases.

In order to guarantee consistency, two checks must be performed: consistency of the union of knowledge bases (since the union of first-order theories or the union of default theories may be inconsistent), and consistency of this union with respect to the integrity constraints. To satisfy the maximality condition the combined knowledge base of a set of Horn programs may have to be transformed into a disjunctive knowledge base. Methods are presented that consider these aspects and construct the maximal consistent combined knowledge base correct with respect to a given set of knowledge bases. Combining knowledge bases may be compared to the problem of updating a knowledge base [BKMS91b].

Work achieved earlier on this subject was extended to consider the case of combining databases described by first-order theories [BKMS91b, BKMS92]. A preliminary paper was written on combining databases in default theories [BKMS91a*] and a journal article was written to complete this work [BKMS94].

In addition, we have addressed the problem of combining databases with priorities. One database may have priority over another database for a particular datum or topic, while other

databases may have priority for other data or topics. A paper was accepted for publication in a journal on this subject [PMS95] in which we solve the problem for propositional databases. A preliminary paper on combining databases in *Datalog* databases was presented at a conference [Pra95], and an extended version of the paper has been submitted for publication in a journal [PM95].

2.3 Disjunctive Deductive Databases.

2.3.1 Overview of Deductive Database Research

Disjunctive deductive databases (*DDDBs*) are concerned with real world situations that are both definite and indefinite. This contrasts with work in relational databases (*RDBs*) and deductive databases (*DDBs*) where all information is definite. In *RDBs* and *DDBs* a fact is either known or derivable and hence *true*, or *false* otherwise. Such databases have no formal mechanism to provide the user with information that is indefinite. Thus, it is not possible to store information of the type, "Smirnov is General of the 5th Russian Air Wing or of the 7th Russian Air Wing."

The state-of-the-art in *DDDBs* before the current grant is described in [FM92b]. During the first year of the grant we extended the theory so that there is now a clear semantics for *DDDBs* that includes *stratified* and *normal DDDBs*. The latter two theories permit negation in the body of rules. The *normal* theory applies to *stable model semantics*. In addition to a clear semantics, we have developed algorithms to store and retrieve data in these databases. The algorithms are based on the *model tree* data structure developed under the grant. A model tree data structure represents completely the minimal models of a *DDDB*.

Two major topics are discussed in this area: bottom-up methods to evaluate disjunctive deductive databases and the view update problem in disjunctive deductive databases.

Bottom-Up Evaluation of Disjunctive Databases. Given a hierarchical, range restricted disjunctive deductive database, an algorithm was developed to compute answers efficiently to queries [FM91a*]. The computation method is based on the development of a model tree which shares minimal models of a disjunctive database. An incremental algorithm is presented which computes the model tree of a hierarchic disjunctive deductive database in one pass through the rules of the database.

During the current grant, we developed a new fixpoint characterization of the minimal models of disjunctive logic programs [FM91]. We proved that applying the operator iteratively characterizes the perfect models semantics of disjunctive stratified logic programs. Given the equivalence between the *perfect models semantics of stratified programs* and *prioritized circumscription*, our fixpoint characterization captures the meaning of the corresponding circumscribed theory. Based on these results, we present a bottom-up evaluation algorithm for disjunctive stratified databases. This algorithm uses the *model-tree* data structure to represent the information contained in the database and to carry on the computation of queries.

We also show [FLMS93] that stable models of logic programs may be viewed as minimal models of programs that satisfy certain additional constraints. To do so, we transform the normal programs into disjunctive logic programs and sets of integrity constraints. We show that the stable models of the normal program coincide with the minimal models of the disjunctive program that *satisfy* the integrity constraints. As a consequence, the stable model

semantics can be characterized using the *Extended Generalized Closed World Assumption* for disjunctive logic programs.

We also note that for disjunctive logic programs the two definitions of integrity constraint satisfaction: *entailment* and *consistency* differ since disjunctive programs have more than one minimal model.

Using this result, we develop a bottom-up algorithm for function-free logic programs to find all stable models of a normal program by computing the minimal models of a disjunctive logic program and checking them for consistency with the integrity constraints. This algorithm uses the *model-tree* abstract data structure developed by Fernández and Minker to compute the minimal models of disjunctive deductive databases. We complement Fernández and Minker's original algorithms by adding a new step where models inconsistent with the integrity constraints are ruled out.

The integrity constraints provide a rationale as to why some normal logic programs have no stable models. Using the minimal models of the program and the set of integrity constraints that are inconsistent with those models, it is possible to determine why a normal program has no stable models. As a result of this work, a journal article appeared on computing stable model semantics [FLMS93]. In addition, two invited conference papers [FM92a, FM92b] were presented. One invited journal article [FM93] was written on the theoretical aspects of disjunctive deductive databases. We present the model, fixpoint and proof semantics for a large class of disjunctive deductive databases. Algorithms are presented in all cases. A Ph.D. thesis was written by Fernández that provides the theory and algorithms for disjunctive deductive database [Fern93]. In addition, since disjunctive deductive databases are, in general, intractable, we have devised an algorithm to handle a tractable subclass of these databases in which every clause in the theory has at most two literals [FKM92].

We have also developed model, proof and fixpoint semantics for extended disjunctive deductive databases [MR93] which combines default negation and classical negation. An invited journal paper was written on this subject [MR94]. Several papers were written on representing, evaluating, and computing in disjunctive databases [FMY95, YFM94, YM93, YM95].

View Update in Disjunctive Databases. The view update problem is the following: given disjunctive facts and rules such that the predicates in the disjunctive facts are distinct from predicates in the heads of rules, how does one enter or delete a predicate or a disjunction of predicates that appears in a rule. The problem is complex as one cannot simply add the predicate that appears in the head of a rule to be a fact; it cannot appear as a fact since the predicates of facts must be distinct from the predicates that appear in the heads of rules. This condition applies not only to deductive databases, but to relational databases as well. In a relational database, if a relation is defined as a view, then it cannot be defined as a relational table at the same time.

Grant, Horty, Lobo and Minker [GHLM92, GHLM93] develop algorithms to insert and delete views in disjunctive deductive databases. They show that there are many ways in which one can enter or delete a predicate defined as a view in a disjunctive deductive database and show that the method they develop is the best with respect to certain criteria. Fernández, Grant and Minker [FGM94] develop a model theoretic approach to the view update problem that generalizes the approach in [GHLM93]. Gryz, Grant and Minker [GGM95] show how, using model trees, to update ground disjunctive databases without deductive rules.

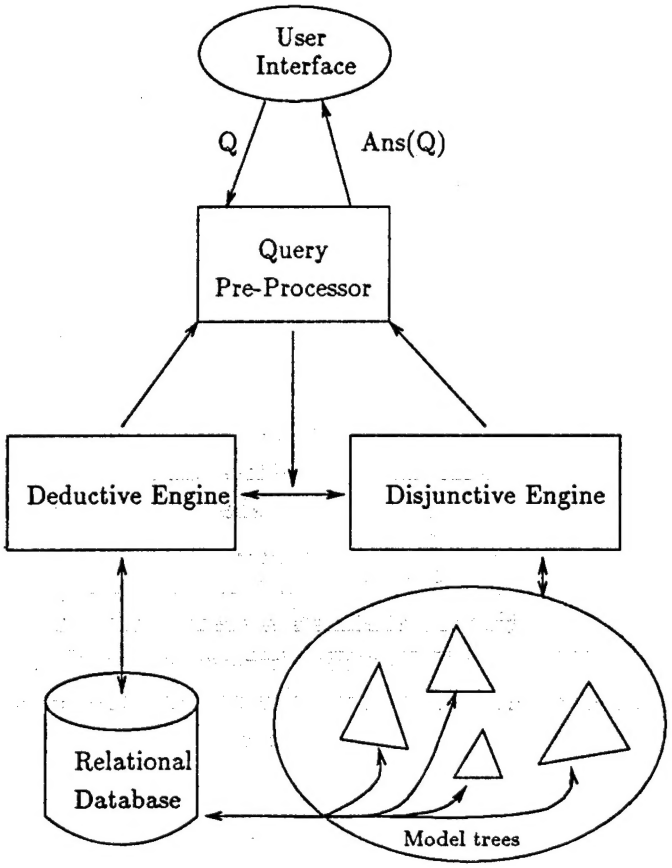

Figure 1: Functional units in DDDBS

In addition to the above work, Grant and Minker [GM92b] wrote an invited paper in which they describe the impact of logic programming on databases. Grant and Minker [GM92a, Min92] wrote articles on deductive databases that were published in encyclopedias.

2.3.2 Disjunctive Deductive Database Prototype

Disjunctive theories are important when data are unknown or uncertain, as is often the case in military intelligence applications. Based upon our theoretical work on disjunctive theories, we have developed an architecture for a disjunctive deductive database system [Fern93] which is shown in Figure 1. We developed a prototype system to experiment with disjunctive deductive databases. An on-line demonstration of this implementation is accessible through WWW at

URL = http://karna.cs.umd.edu:3264/

In what follows, we describe each component of the architecture shown in Figure 1.

Relational Database. This part of the prototype consists of all the relational tables containing definite facts.

Model trees. The semantics of a set of disjunctive facts in the database is represented by the set of its minimal models. A disjunction of the form $a \vee b$ forces every minimal model to contain either a or b.

Dealing with indefinite data is, in general, computationally intractable. The source of intractability of a DDDB lies in the necessity of representing and then working with all (minimal) models of the database. One efficient way to implement minimal models is to use a structure called *model tree* developed by Fernández and Minker in [FM91a*, Fern93]. A model tree allows structure sharing so that the same atom need not be listed separately for each model. Nodes in such a tree structure represent the ground atoms of the database. Each branch (a path from the root to a leaf node) represents a model.

This representation can be improved further by defining the concept of a *model forest* [Fern93]. A model forest is a clustered representation of a model tree. Instead of representing all minimal models of the database in a single tree, it is split into smaller trees that do not share atoms. With the model forest representation, any query containing predicates from a single tree, needs to be evaluated in that tree only.

This clustered representation of the database may reduce dramatically the space needed for storing the database as well as the time needed for query processing. For example, in the extreme case where a database contains n independent disjunctions each containing 2 disjuncts, it can be represented as n 3-node (2 nodes for the disjuncts plus one node for a root) trees. On the other hand, if one were to represent the same database in one tree, it would have $O(2^n)$ nodes!

Query Pre-processor, Deductive Engine, and Disjunctive Engine. All predicates in the DDDB are partitioned into two classes: *extensional* and *intensional.* The former are used to represent facts (definite or indefinite) and corresponds to facts and the latter represent information that can be inferred using the deductive capability of the *DDDB* (i.e., information that occur in heads of rules). Another partition (orthogonal to the first) is between *definite* and *indefinite* predicates. A predicate p is indefinite if and only if

- p occurs in a disjunctive fact, or
- p occurs in the head of a disjunctive rule, or
- p occurs in the head of any rule whose body contains an indefinite predicate.

The fact that a predicate is indefinite does not imply that all its instances represent indefinite information. For example, if $teach(jarek, philosophy)\ OR\ teach(jarek, ai)$ is a disjunctive fact in our database, *teach* is an indefinite predicate. There could be, however, other facts with this predicate, e.g., $teach(michael, ai)$, which are definite. Definite facts are stored explicitly in the relational database and indefinite facts (disjunctions) are stored implicitly as model trees.

An answer to a query $\leftarrow Q(\vec{X})$ is a substitution θ for the variables $\vec{X}$ appearing in Q in such a way that $Q\theta$ is *true* in the database. Given a query, our prototype retrieves all possible such substitutions or answers. In what follows, we sometimes call these substitutions "bindings".

Each query sent to our database is processed in the following way.

1. The query pre–processor flattens the query; i.e., finds all alternative conjunctions of *extensional* atoms that are equivalent to the original query. Without disjunctive rules, flattenings can be produced efficiently by top-down PROLOG evaluation.

2. Next, each of the flattened queries (which at this point contains only extensional predicates) $Q_i, 1 \leq i \leq k$, where k is the number of flattenings) is separated into two parts: a definite and an indefinite part. Thus, each of Q_i's has the form $\leftarrow p'_1, ..., p'_n, q'_1, ..., q'_m$ where the p''s are definite and the q''s are indefinite predicates (for simplicity we do not indicate the arguments of the predicates).

3. Depending on the values of n and m, the query is sent to the deductive engine, to the disjunctive engine or to both.

   - If $n > 0,\ m = 0$, i.e., a query contains only definite predicates:
     The query is sent to the deductive engine and then translated into SQL and sent to the relational database for processing.
   - If $n = 0,\ m > 0$, i.e., a query contains only indefinite predicates:
     The query is sent to the disjunctive engine for processing. The disjunctive engine retrieves answers from the model trees.
     The way the disjunctive engine processes the query is by checking, for every model of a respective tree, whether there is a binding for which the query is true in that model, and then building a disjunction of all such bindings.
     Clearly, the tree lookup should not be executed by brute force. Each tree should be accompanied by an indexing scheme which would support fast search for all atoms in a given model, as well as for all models which contain a given atom.
   - If $n, m > 0$, i.e., a mixed query:
     If the definite and the indefinite part do not share variables, each part can be processed independently (in the same fashion as described above) and the retrieved partial answers are concatenated into a (complete) answer. If, however, the two parts of the query share variables, they have to be processed serially. The definite part of the query, i.e., $\leftarrow p'_1, ..., p'_n$, is sent to a relational database and the appropriate answers are retrieved. These answers represent alternative substitutions that need to be made (by the disjunctive engine) to the variables in the indefinite part of the query. Both the answers retrieved from the relational database and from the indefinite part of the query are sent to the disjunctive engine for processing.

Complexity Considerations. Although dealing with indefinite data is, in general, computationally intractable, by clustering the indefinite data, it is possible to answer queries to indefinite data in times that are comparable to searching over relational databases. This happens when the clusters are reasonably small. In Figure 2, we show the limit of the size of clusters of the database that permits reasonable efficiency of query processing. By "reasonable," we mean that a DDDB should spend no more time processing the indefinite part of the query as it does processing the definite part. Figure 2 shows points at which the two parts of the equation are equal, i.e., when the complexity of finding all answers to the definite

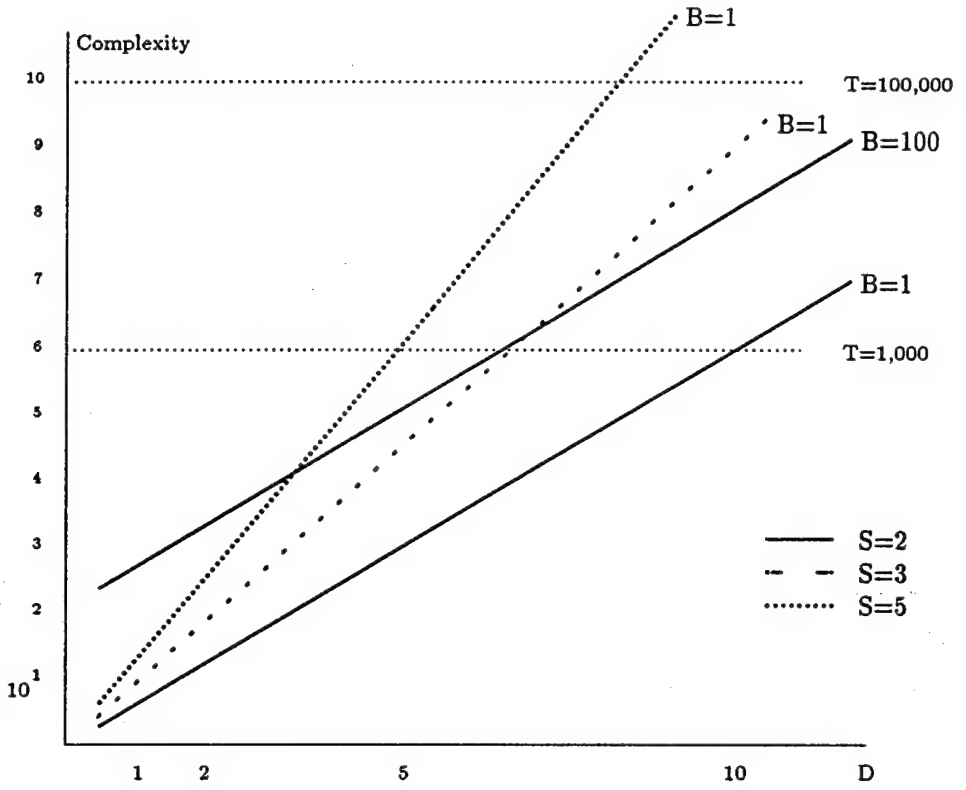

Figure 2: A comparison of complexity: definite vs. indefinite part of the query

part is equal to that of the indefinite part. The complexity of finding all answers to a mixed query, which consists of a conjunction of definite and indefinite atoms is given below.

Let the query be: $\leftarrow p_1, ..., p_l, q_1, ..., q_m$, where the p's are definite and the q's are indefinite predicates. We use the following notation:

- T = Average number of tuples in table
- D = Average number of disjunctions per cluster
- B = Average number of bindings passed from the definite to the indefinite part of the query
- S = Average number of atoms in disjunctions

If the p's and q's share variables the complexity of finding all answers to the query can be estimated as:

$$Complexity \;=\; B * S^{m*D} \;+\; T^l$$

Figure 2 shows that if the number of disjunctions per cluster is, say 6, and the average number of bindings passed from the definite part of the query to the indefinite part is 100,

and the average number of tuples in a relational table is 1,000, then the complexity of finding all answers to a query $\leftarrow p_1, p_2, q_1, q_2$ is the same for retrieving data from the relational tables and the disjunctive data. Hence, in this case the search for answers is no more complex for the disjunctive part as for the relational part of the query.

2.4 Extensions to the Semantics of Logic Programs.

A research monograph, *Foundations of Disjunctive Logic Programming* by J. Lobo, J. Minker and A. Rajasekar [LMR92], was completed. The monograph has been used as a course for graduate students in computer science. Each chapter motivates the material in it and illustrates definitions, theorems and algorithms with examples. Each chapter, with the exception of Chapter 1, contains a variety of exercises enabling the student to test whether or not the material is understood. Hints are provided on most exercises. In addition, each chapter contains background material and historical references.

The book contains 10 chapters, whose titles are:

Chapter 1: Introduction and Background

Chapter 2: Definitions and Terminology

Chapter 3: Declarative Semantics

Chapter 4: Proof Theory

Chapter 5: Negation

Chapter 6: Weak Negation

Chapter 7: Normal Logic Programs

Chapter 8: Proof Theory: Normal Programs

Chapter 9: Disjunctive Deductive Databases

Chapter 10: Applications

In addition to the book, a new semantics was developed for normal disjunctive logic programs [BLM91], termed WF^3. The semantics extends the Generalized Disjunctive Well Founded Semantics ($GDWFS$), reported upon in the 1991 progress report [BLM92]. Model, proof and fixpoint semantics have been developed for the WF^3.

In [LMR91*], alternative theories developed for logic programs and disjunctive logic programs are described. In an invited lecture, [Min94], an overview is given of work in disjunctive logic programming. In an invited paper in honor of Alan Robinson [MLR91], the major results obtained in disjunctive databases are enumerated.

We have developed fixpoint, model, and proof semantics for a wide class of extended logic programs and disjunctive logic programs [MR93]. An extended disjunctive logic program has two kinds of negation: classical and negation-by-default. Literals may appear in the head of a clause, while literals and negated-by-default literals may appear in the body of a clause. An invited journal article [MR94] has been written on this subject. The work was supported primarily by the NSF.

2.5 Parallel Logic Programming.

Work on parallel logic programming was not one of the topics of research on the current grant. However, work on parallel logic programming was initiated based on earlier grants by the AFOSR. The work described below was supported by the earlier grants and a grant from the NSF. It is reported upon here as it is a consequence of support from the AFOSR.

A logic programming system augmented with constraint processing, data storage, and data manipulation capabilities forms the basis for a knowledge base system. Both a run-time and a compile-time approach to using integrity constraints in logic programming systems to identify and eliminate unproductive search activity have been implemented within an existing parallel logic programming system, PRISM. The extended system provides the basis for a series of experiments which demonstrate that significant classes of knowledge representation domains can use integrity constraints effectively. In [GGL+93] we show that using constraints to process a query can reduce search space and response time. Furthermore, we show that in certain cases the compile-time approach reduces response time much more than the run-time approach.

In [Lin92], a new scheduling scheme is proposed which directs processors to share the search space according to universal task distribution rules obeyed by all processors involved. Load balancing is achieved by altering the shape of a search tree to remove the so-called structural imbalance, and following a statistically even distribution rule. A condition for task distribution is derived which minimizes the average parallel runtime. We present data showing the effectiveness of the proposed scheme. Simulation results from benchmark programs that can be found in the literature demonstrate that the method is able to treat programs efficiently that render mostly fine-grained parallel tasks under a typical existing scheduler. The peak speed-up factors with the proposed technique exceed by a substantial margin that achieved by Aurora Parallel Prolog on the same set of benchmarks.

Dynamic load balancing is the key to achieving high performance as well as maximum utilization of processors in the parallel execution of logic programs, in which parallelism is defined implicitly. Lin, in his thesis [Lin92b], investigates issues pertinent to scheduling and load balancing in the parallel execution of logic programs in *a distributed memory system.* A paper on this subject has appeared in the proceedings of a conference [Lin92a]. Several task scheduling strategies are proposed and their performance evaluated through analytical and experimental (simulation and emulation) studies. It is shown that there exists a balance between task scheduling overhead and balancing load distribution. The goal for a scheduler is to reach the balance so as to maximize the utilization of a parallel computer system. This objective is approached from two directions:

- minimizing scheduling overhead under the proposition of balancing load distribution;
- balancing load distribution under the proposition of no communication overhead.

The thesis demonstrates that a combined method yields substantial improvement over what has been achieved by existing techniques. Results from the thesis provide a deeper insight into the role task scheduling plays in the execution of parallel logic programs. Evidence obtained shows that a large scale multiprocessor can be used efficiently by a parallel logic programming system given an efficient task scheduling mechanism. The method was implemented on a SUN SPARCstation. Results for 30 problems are discussed and contrasted with those obtained in the literature.

3 Summary of Publications

The following is a list of journal, book chapters, conference papers, and technical reports published during the current grant with partial support from the AFOSR. A summary of the number of papers written or presented is as follows:

- Research Monograph: 1
- Journal articles: 11
- Invited Journal article: 5
- Book chapters: 1
- Conference proceedings (refereed): 8
- Conference proceedings (invited): 3
- Encyclopedia articles: 2
- Technical reports (not published): 5
- Workshop Presentations: 3
- Ph.D. theses: 3
- Ph.D. theses (in progress): 1

Papers referenced with an asterisk (*) were published prior to the award of the grant and are not included in the above count.

4 Graduate Students Supported

The following graduate students have been supported under the grant:

- José Alberto Fernández
- Terry Gaasterland
- Parke Godfrey
- John Guthrie
- Jarek Gryz
- Zahid Khandaker
- Yuan Liu
- Shekar Pradhan
- Carolina Ruiz

References

[BKMS91a*] C. Baral, S. Kraus, J. Minker, and V.S. Subrahmanian. "Default Logics Combining and Updating Default Theories: Preliminary Report," *Bar-Ilan Symposium on the Foundations of Artificial Intelligence (BISFAI-91), Ramat Gan, Israel, June 16-19, 1991.*

[BKMS91b] C. Baral, S. Kraus, J. Minker, and V.S. Subrahmanian. "Combining Knowledge Bases Consisting of First Order Theories," *6th International Symposium on Methodologies for Intelligent Systems (ISMIS'91)*, Lecture Notes in Artificial Intelligence (542), Z.W. Ras and M. Zemankova (Eds.), Charlotte, NC, 92-101, Oct. 1991.

[BKMS92] C. Baral, S. Kraus, J. Minker, and V.S. Subrahmanian. "Combining Knowledge Bases Consisting of First Order Theories," *Computational Intelligence, Vol. 8, No. 1, 45-71, 1992.*

[BKMS94] C. Baral, S. Kraus, J. Minker, and V.S. Subrahmanian. Combining default logic databases. *Journal of International Information Systems*, 1994. To appear.

[BLM91] C. Baral, J. Lobo, and J. Minker. WF^3: A semantics for negation in normal disjunctive logic programs. In *6th International Symposium on Methodologies for Intelligent Systems (ISMIS'91), Lecture Notes in Artificial Intelligence (542)*, pages 459–468, Charlotte, NC, October 1991.

[BLM92] C. Baral, J. Lobo, and J. Minker. Generalized disjunctive well-founded semantics for logic programs. *Annals of Mathematics and Artificial Intelligence*, 5(2-4):89–131, May 1992.

[DG92] B. J. Dorr and T. Gaasterland. "Using a Temporal Database as the Basis for Natural Language Generation and Translation," *Advanced Information Processing and Analysis Symposium*, March, 1992.

[Fern93] J.A. Fernández. Disjunctive Deductive Databases. Ph.D. thesis, University of Maryland, CS-TR-3208, UMIACS-TR-94-6.

[FGM94] José Alberto Fernández, John Grant, and Jack Minker. Model theoretic approach to view updates in deductive databases. Technical Report CS–TR–3335, Department of Computer Science, University of Maryland, College Park, MD 20742, 1994.

[FKM92] J.A. Fernández, Z.A. Khandakar, and J. Minker. A tractable class of disjunctive deductive databases. In *Proc. Workshop on Deductive Databases, Joint International Conference and Symposium on Logic Programming (JICSLP'92)*, Washington, D.C., Nov. 1992.

[FLMS93] J.A. Fernández, J. Lobo, J. Minker, and V.S. Subrahmanian. Disjunctive LP + integrity constraints = stable model semantics. *Annals of Mathematics and Artificial Intelligence*, 8(3–4):449–474, 1993.

[FM91] J.A. Fernández and J. Minker. Computing perfect models of disjunctive stratified databases. In *ILPS'91 Workshop on Disjunctive Logic Programming*, November 1991.

[FM92a] J.A. Fernández and J. Minker. Disjunctive deductive database. In *3rd International Conference on Logic Programming and Automated Reasoning*, pages 332–356, July 1992. Invited Paper.

[FM92b] J.A. Fernández and J. Minker "Semantics of Disjunctive Deductive Databases", *4th International Conference on Database Theory*, Berlin, Germany, Oct. 1992, 21-50. (Invited Paper).

[FM93] J.A. Fernández and J. Minker. Theory and algorithms for disjunctive deductive databases. *Programmirovanie*, N 3:5–39, 1993. (also appears as University of Maryland Technical Report,CS-TR-3223, UMIACS-TR-94-17,1994. Invited Paper in Russian).

[FM91a*] J.A. Fernández and J. Minker. "Bottom-up Evaluation in Disjunctive Deductive Databases," *Proceedings of the International Conference on Logic Programming, Paris*, June 91.

[FMY95] J.A. Fernández, J. Minker, and A. Yahya. Computing perfect and stable models using ordered model trees. *Journal of Computational Intelligence*, Vol. 11, No.1, 89-112, 1995.

[GM90*] A. Gal and J. Minker. "Producing Cooperative Answers in Deductive Databases," In Patrick St. Dizier and S. Szpakowicz, editors, *Logic and Logic Grammer for Language Processing*, pages 223–254. Ellis Horwood Ltd., 1990.

[GHLM92] J. Grant, J. Horty, J. Lobo, and J. Minker. "Updates in Disjunctive Deductive Databases," *Second International Workshop in Artificial Intelligence and Mathematics, Fort Lauderdale, CA., January 1992.*

[Gaa92a] T. Gaasterland. Cooperative explanation in deductive databases. In *AAAI Spring Symposium on Cooperative Explanation*, pages 25–27, March 1992.

[Gaa92b] T. Gaasterland. *Generating Cooperative Answers in Deductive Databases.* PhD thesis, University of Maryland, College Park, MD 20742, 1992.

[GGL+93] T. Gaasterland, M. Giuliano, A. Litcher, Y. Liu, and J. Minker. Using integrity constraints to control search in knowledge base systems. *International Journal of Expert Systems*, 6(4):447–487, 1993.

[GGM91*] T. Gaasterland, P. Godfrey, and J. Minker. Relaxation as a platform for cooperative answering. In *International Workshop on Non-Standard Answers and Non-standard Queries*, pages 1–3, Toulouse, France, July 1991.

[GGM92a] T. Gaasterland, P. Godfrey, and J. Minker. An overview of cooperative answering. *Journal of Intelligent Information Systems*, 1(2):123–157, 1992. (invited paper).

[GGM92b] T. Gaasterland, P. Godfrey, and J. Minker. Relaxation as a platform for cooperative answering. *Journal of Intelligent Information Systems*, 1(3/4):193–321, 1992.

[GGMN92c] T. Gaasterland, P. Godfrey, J. Minker, and L. Novik. A cooperative answering system. In Andrei Voronkov, editor, *Proceedings of the Logic Programming and Automated Reasoning Conference*, Lecture Notes in Artificial Intelligence 624, pages 478–480. Springer-Verlag, St. Petersburg, Russia, July 1992.

[GMN94] P. Godfrey, J. Minker, and L. Novik. An architecture for a cooperative database system. In Witold Litwin and Tore Risch, editors, *Proceedings of the First International Conference on Applications of Databases*, Lecture Notes in Computer Science 819, pages 3–24. Springer Verlag, Vadstena, Sweden, June 1994. (Invited Paper.)

[Godf95] P. Godfrey An Architecture and Implementation for a Cooperative Database System PhD thesis, University of Maryland, College Park, MD 20742, 1995 (In Progress).

[Godf94] P. Godfrey Minimization in Cooperative Response to Failing Database Queries University of Maryland Technical Report CS-TR-3348 and UMIACS-TR-94-108, 1994.

[GHLM93] J. Grant, J. Horty, J. Lobo, and J. Minker. View updates in stratified disjunctive databases. *Journal Automated Reasoning*, 11:249–267, March 1993.

[GM92a] J. Grant and J. Minker. Deductive database systems. In *Encyclopedia of Artificial Intelligence*, pages 320–329. 1992.

[GGM95] J. Grant, J. Gryz, and J. Minker Updating disjunctive databases via model trees. *University of Maryland Technical Report, CS-TR-3407, UMIACS-TR-95-11.* February 1995.

[GM92b] J. Grant and J. Minker. The impact of logic programming on databases. *Communications of the ACM*, 35(3):66–81, March 1992. (Invited Paper.)

[Lin92] Z. Lin. Self-organizing task scheduling for parallel execution of logic programs. In *Proceeding of the 1992 Conference on Fifth Generation Computer Systems*, Japan, 1992.

[Lin92a] Z. Lin. A Distributed Load Balancing Scheme for Parallel Logic Programming 21st International Conference on Parallel Processing.

[Lin92b] Z. Lin. "Task Scheduling for Parallel Execution of Logic Programs". *Ph.D. Thesis, Department of Computer Science, University of Maryland, May 1992.*

[LMR91*] J. Lobo, J. Minker, and A. Rajasekar. On general disjunctive logic programs. In *Intelligent Systems*, pages 170–199, 1991.

[LMR92] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming.* MIT Press, 1992.

[Min94] J. Minker. An overview of disjunctive logic programming. *Annals of Mathematics and Artificial Intelligence.* Vol. 12 (1994), Nos. 1,2, 1–24, December, 1994. (Invited Paper.)

[Min92] J. Minker "Deductive Database" *Encyclopedia of Computer Science, A. Ralston, Editor,* 320-328, 1992.

[MLR91] J. Minker, J. Lobo, and A. Rajasekar. Theory of disjunctive logic programming. In J.-L. Lassez, editor, *Computational Logic: Essays in honor of Alan Robinson*, pages 613–639. MIT Press, 1991.

[MR93] J. Minker and C. Ruiz. On extended disjunctive logic programs. In J. Komorowski and Z.W. Ras, editors, *Proceedings of 7th Intl. Symposium - ISMIS'93*, pages 1–18, Trondheim, Norway, June 1993. Springer-Verlag. (Invited Paper.)

[MR94] J. Minker and C. Ruiz. Semantics for disjunctive logic programs with explicit and default negation. *Fundamenta Informaticae*, 20(3/4):145–192, 1994. Anniversary Issue edited by H. Rasiowa. (Invited Paper.)

[Pra95] S. Pradhan. Combining datalog databases using priorities. In *Advances in Data Management '94*, pages 355–375. Tata-McGraw Hill, India, 1995.

[PM95] S. Pradhan and J. Minker. Combining datalog database with priorities. Technical Report, University of Maryland, CS-TR-3470, UMIACS-TR95-58, May 1995.

[PMS95] S. Pradhan, J. Minker, and V.S. Subrahmanian. Combining databases with prioritized information. *Journal of Intelligent Information Systems*, 4(3):231–260, May 1995.

[YFM94] A. Yahya, J.A. Fernández, and J. Minker. Ordered model trees: A normal form for disjunctive deductive databases. *Journal of Automated Reasoning*, 13(1):117–143, August 1994.

[YM93] A. Yahya and J. Minker. Representations for disjunctive deductive database. Technical Report CS-TR-3111, UMIACS-TR-93-70, University of Maryland, 1993.

[YM95] A. Yahya and J. Minker. Query evaluation in partitioned disjunctive deductive databases. *International Journal of Intelligent and Cooperative Information Systems*, 1995. To appear.